

Programujeme STM32

bez knihoven

Ing. Vojtěch Skřivánek

```
TIM2->CR2 |= TIM_CR2_MMS_1;
```

```
TIM2->CR1 |= TIM_CR1_GEN;
```

```
DMA1_Channel5->CCR |= (DMA_CCR_MINC | DMA_CCR_EN);
```

```
DMA1_Channel5->CNDTR = 1;
```

```
DMA1_Channel5->CR = 0;
```

```
DMA1_Channel5->CCR = 0;
```

```
RCC->IOPBENR |= (1 << PA);
```

```
GPIOA->MODER = 0x01;
```

```
GPIOA->MODER = 0x01;
```

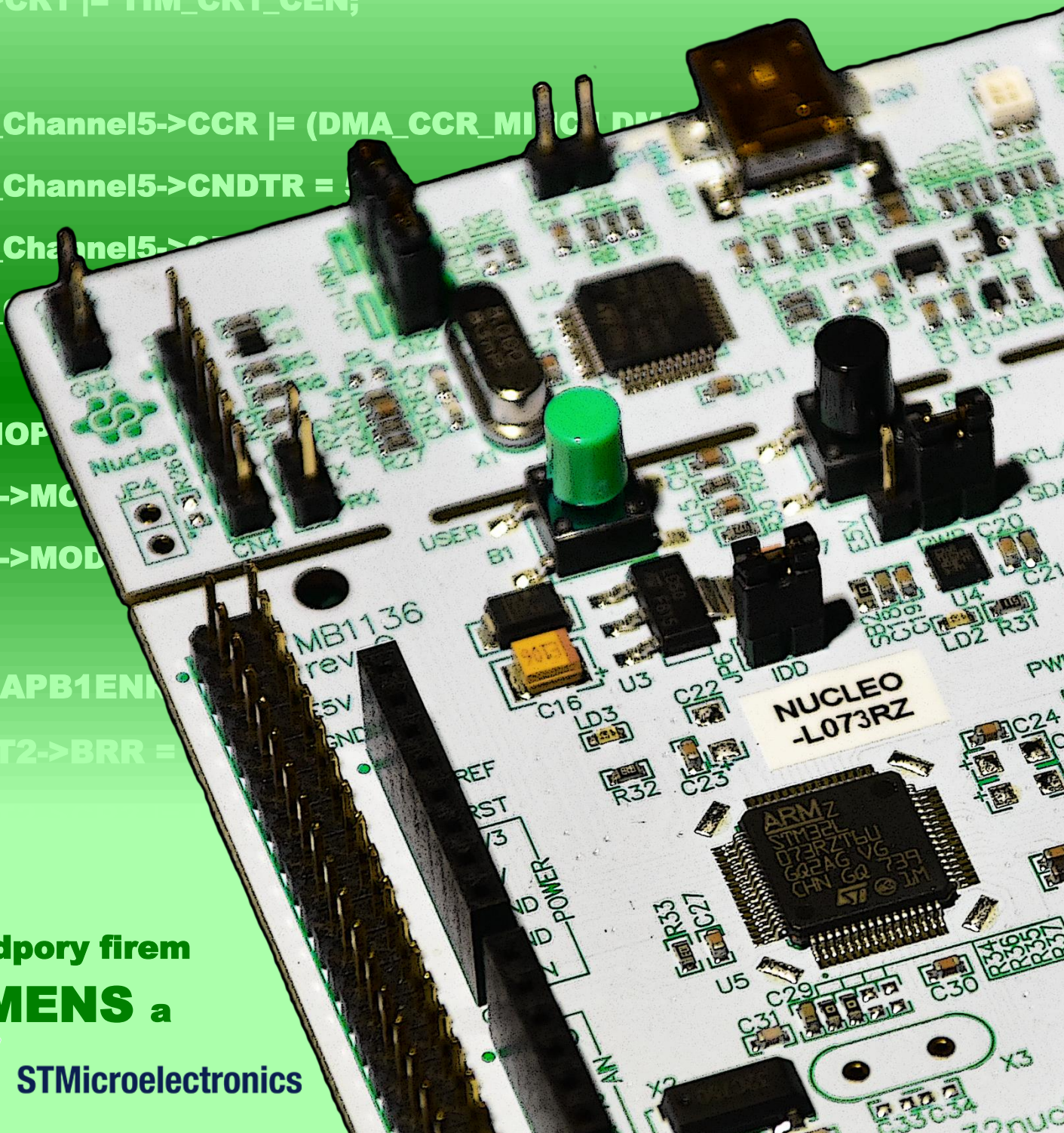
```
RCC->APB1ENR |= (1 << USART2);
```

```
USART2->BRR = 0x00000000;
```

Za podpory firem

SIEMENS a

 STMicroelectronics



Poděkování

Děkuji firmám

SIEMENS

a



za podporu při psaní této knihy.

Obsah

1	Úvod	1
1.1	Motivace knihy	1
1.2	Struktura knihy	3
1.3	Fonty textu	4
2	Vývojové nástroje	5
2.1	Vývojová deska	6
2.2	Vývojové prostředí	7
2.3	Dokumentace	7
2.4	Shrnutí	8
3	Založení projektu	9
4	Vstupně/výstupní piny	15
4.1	Tlačítkem rozsvícená LED	15
4.1.1	Práce s registry při tvorbě programu	16
4.1.1.1	Nalezení symbolického názvu registru	16
4.1.1.2	Nalezení symbolického názvu bitu	16
4.1.1.3	Zápis do registru	17
4.1.2	Tvorba programu	17
4.1.3	Porovnání	18
5	Přerušení	21
5.1	Tlačítkem rozsvícená LED pomocí přerušení	22
5.1.1	Nastavení periférií	22
5.1.1.1	LED	22
5.1.1.2	EXTI	22
5.1.2	Funkce periférií	24
5.1.2.1	LED - přepínání	24
5.1.3	Obsluha přerušení	24
5.1.4	Tvorba hlavního programu	26
5.1.5	Zapojení	26
5.1.6	Porovnání	27
6	Časovač	29
6.1	Blikání pomocí časovače s přerušením	29
6.1.1	Nastavení periférií	29
6.1.1.1	RCC - hodinový signál	29
6.1.1.2	TIM	31
6.1.2	Funkce periférií	32
6.1.2.1	TIM - start	32

6.1.3	Obsluha přerušení	32
6.1.4	Tvorba programu	33
6.1.5	Porovnání	34
6.2	Blikání pomocí OC režimu	35
6.2.1	Nastavení periférií	35
6.2.1.1	TIM	35
6.2.2	Tvorba programu	37
6.2.3	Provnání	37
6.3	Blikání s nastavením délky pomocí IC režimu	38
6.3.1	Nastavení periférií	38
6.3.1.1	TIM3	38
6.3.2	Funkce periférií	39
6.3.2.1	TIM3 start	39
6.3.3	Obsluha přerušení	40
6.3.4	Tvorba programu	41
6.3.5	Zapojení	41
6.3.6	Rozbor programu	42
6.3.7	Porovnání	42
6.4	Nastavení jasu LED pomocí PWM režimu	43
6.4.1	Nastavení periférií	43
6.4.1.1	TIM	43
6.4.2	Funkce periférií	44
6.4.2.1	TIM2 - změna střídy	44
6.4.3	Tvorba programu	45
6.4.4	Rozbor programu	45
6.4.5	Porovnání	46
6.5	Časovač spuštěný čítačem vnějších událostí	47
6.5.1	Nastavení periférií	48
6.5.1.1	TIM2	48
6.5.1.2	TIM21	49
6.5.2	Funkce periférií	50
6.5.2.1	TIM21 - start	50
6.5.3	Tvorba programu	50
6.5.4	Zapojení	51
6.5.5	Porovnání	51
7	UART	53
7.1	LED rozsvícená povelom z počítače	53
7.1.1	Převodník sériové komunikace	53
7.1.2	Nastavení periférií	54
7.1.2.1	UART	54
7.1.3	Funkce periférií	55
7.1.3.1	UART - odesílání dat	55
7.1.3.2	UART - příjem dat	56
7.1.3.3	UART - callback příjmu	58
7.1.4	Obsluha přerušení	59
7.1.5	Tvorba programu	60
7.1.6	Porovnání	60

8	ADC	61
8.1	Přerušované měření dvou kanálů	61
8.1.1	Nastavení periférií	61
8.1.1.1	ADC	61
8.1.1.2	UART	63
8.1.2	Funkce periférií	63
8.1.2.1	ADC - start	63
8.1.2.2	UART – odešli data v blokujícím režimu	64
8.1.3	Obsluha přerušení	64
8.1.4	Tvorba programu	65
8.1.5	Zapojení	66
8.1.6	Porovnání	67
8.2	Jednotlivé měření dvou kanálů s externím spouštěním	68
8.2.1	Nastavení periférií	68
8.2.1.1	ADC	68
8.2.1.2	EXTI	70
8.2.2	Obsluha přerušení	71
8.2.3	Tvorba programu	72
8.2.4	Zapojení	72
8.2.5	Porovnání	73
9	DAC	75
9.1	9.1 Nastavení jasu LED pomocí DAC	75
9.1.1	Nastavení periférií	75
9.1.1.1	DAC	75
9.1.2	Funkce periférií	76
9.1.2.1	DAC - nastavení výstupní hodnoty	76
9.1.3	Tvorba programu	76
9.1.4	Porovnání	77
10	SPI	79
10.1	Obousměrná SPI komunikace Master<->Slave	79
10.1.1	Nastavení periférií	79
10.1.1.1	SPI1	79
10.1.1.2	SPI2	81
10.1.2	Funkce periférií	82
10.1.2.1	SPI1 – vysílání a příjem v blokujícím režimu	82
10.1.2.2	SPI2 – vysílání a příjem v neblokujícím režimu	82
10.1.3	Obsluha přerušení	84
10.1.4	Tvorba programu	84
10.1.5	Zapojení	85
10.1.6	Rozbor programu	85
10.1.7	Porovnání	86
11	I2C	87
11.1	Jednosměrná komunikace Master->Slave a Master<-Slave	87
11.1.1	Nastavení periférií	88
11.1.1.1	I2C1	88
11.1.1.2	I2C3	89
11.1.2	Funkce periférií	90
11.1.2.1	I2C1 - vysílání	90

11.1.2.2	I2C1 - příjem	92
11.1.2.3	I2C3 - vysílání	93
11.1.2.4	I2C3 - příjem	94
11.1.3	Obsluha přerušení	95
11.1.3.1	I2C1	95
11.1.3.2	I2C3	96
11.1.4	Tvorba programu	97
11.1.5	Zapojení	98
11.1.6	Rozbor programu	98
11.1.7	Porovnání	100
12	DMA	101
12.1	DMA přenos z paměti do paměti	101
12.1.1	Nastavení periférií	101
12.1.1.1	DMA	101
12.1.2	Funkce periférií	102
12.1.2.1	DMA - start	102
12.1.3	Obsluha přerušení	103
12.1.4	Tvorba programu	103
12.1.5	Rozbor programu	104
12.1.6	Porovnání	104
12.2	DMA přenos z periferie do paměti a naopak	105
12.2.1	Nastavení periférií	105
12.2.1.1	UART	105
12.2.1.2	DMA5	106
12.2.1.3	DAC	107
12.2.1.4	DMA4	108
12.2.1.5	TIM	108
12.2.2	Funkce periférií	109
12.2.2.1	DMA5 - start	109
12.2.2.2	DMA4 - start	109
12.2.2.3	TIM - start	109
12.2.3	Tvorba programu	110
12.2.4	Porovnání	110
12.3	DMA přenos z periferie do periferie	111
12.3.1	Nastavení periférií	111
12.3.1.1	ADC	111
12.3.1.2	DMA1	113
12.3.2	Tvorba programu	114
12.3.3	Zapojení	115
12.3.4	Porovnání	116
13	Závěr	117

Kapitola 6

Časovač

Časovač, zejména pokud se jedná a jeho rozšířenou verzi, je velice komplexní periferie.

Námi využitý časovač má 2 řídicí registry, které však skoro nevyužijeme. V praxi se pracuje hlavně s konfiguračními registry jako je registr nastavení předděličky, Autoreload registr, anebo porovnávací registr. Časovač má většinou více kanálů, proto jsou některé tyto konfigurační registry duplikovány pro každý kanál zvlášť. To samé platí pro některé bity v jednotlivých konfiguračních nebo stavových registrech.

6.1 Blikání pomocí časovače s přerušením

V prvním příkladu, který využívá časovače v základním režimu, si ukážeme, jak nastavit volně běžící časovač tak, aby každou sekundu vyvolal přerušení. V obsluze přerušení změníme logickou úroveň výstupního pinu, čímž dojde k zhasnutí či rozsvícení LED nacházející se na *Nucleo* desce s přesně definovaným intervalem jedné sekundy.

Navíc si v tomto příkladu ukážeme, jak můžeme aktivovat vysokorychlostní oscilátor integrovaný v čipu. Dále spustíme fázový závěs (*PLL - Phase-Locked Loop*), kterým budeme systémový hodinový signál upravovat na požadovanou frekvenci 32 MHz. To je maximální frekvence, se kterou kontroler dokáže pracovat.

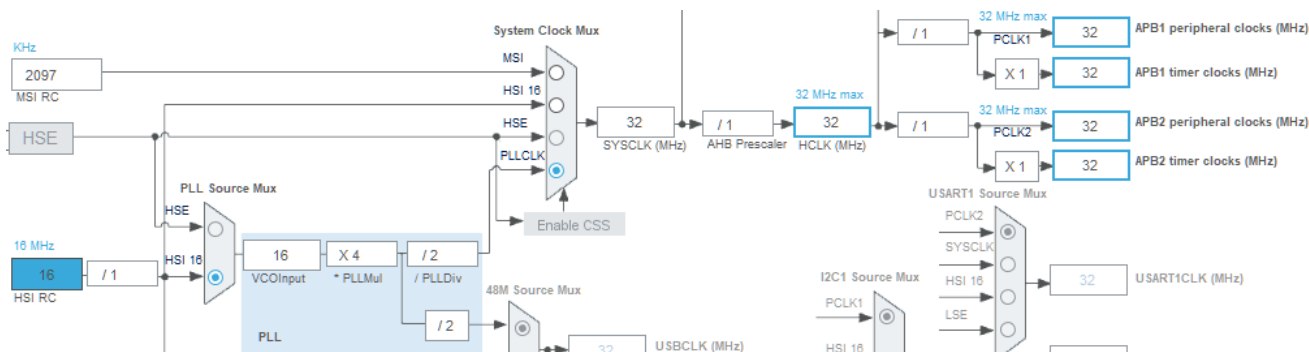
6.1.1 Nastavení periferií

6.1.1.1 RCC - hodinový signál

Nejprve si pro náš program připravíme zdroj hodinového signálu. V tomto příkladu budeme poprvé více pracovat s periferií *RCC* (*Reset and Clock Controller*).

Tato periferie umožňuje zapínat, volit, nastavovat a upravovat zdroje hodinového signálu.

Na obrázku z grafického konfiguratůru vidíme požadovaný výsledek. Chceme nastavit systémový hodinový signál na maximální frekvenci 32 MHz. Toho můžeme docílit tak, že aktivujeme interní vysokorychlostní oscilátor, který připojíme k fázovému závěsu. V něm kmitočet nejprve vynásobíme čtyřikrát a poté vydělíme dvěma. Výstup fázového závěsu pak zvolíme jako zdroj systémového hodinového signálu.



Následující program postupně provede přesně to, co jsme popsali výše.

```
void RCC_init()
{
    // zapne vnitřní vysokorychlostní oscilator (HSI - 16 MHz)
    RCC->CR |= RCC_CR_HSION;

    // cekej, dokud není HSI připraven
    while (!(RCC->CR & RCC_CR_HSIRDY));

    // nastaví násobičku PLL na 4 (16 MHz * 4 = 64 MHz)
    // a děličku PLL na 2 (64 MHz / 2 = 32 MHz)
    RCC->CFGR |= (RCC_CFGR_PLLMUL4 | RCC_CFGR_PLLDIV2);

    // povolí PLL
    RCC->CR |= RCC_CR_PLLON;

    // cekej, dokud není PLL připraven
    while (!(RCC->CR & RCC_CR_PLLRDY));

    // nastaví čas přístupu do FLASH
    FLASH->ACR |= FLASH_ACR_LATENCY;

    // použije PLL jako zdroj systémových hodin
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    // cekej, dokud není použit PLL jako systémové hodiny
    while(!(RCC->CFGR & RCC_CFGR_SWS_PLL));
}
```

Nejprve je aktivován vysokorychlostní oscilátor. Následující smyčka čeká, až bude ustálen. V následujícím kroku dojde k nastavení násobičky a děličky fázového závěsu, který se poté spustí. Opět se čeká na jeho přípravu.

Propojení vysokorychlostního oscilátoru a fázového závěsu je výchozí nastavení kontroleru. To však neplatí o nastavení zdroje systémového hodinového signálu, kterým je vícerychlostní vnitřní oscilátor. Proto je nutné toto nastavení změnit.

Než to ale provedeme, je nutné změnit dobu přístupu do paměti kontroleru. Jelikož po přepnutí zdroje signálu bude frekvence 32 MHz, bylo by čtení z paměti příliš rychlé. O tom se dočteme v příslušné kapitole manuálu kontroleru.

Table 12. Link between master clock power range and frequencies

Name	Power range	Maximum frequency (with 1 wait state)	Maximum frequency (without wait states)
Range 1	1.65 V - 1.95 V	32 MHz	16 MHz
Range 2	1.35 V - 1.65 V	16 MHz	8 MHz
Range 3	1.05 V - 1.35 V	4.2 MHz	4.2 MHz

Tabulka rozsahů napájecích napětí a maximálních frekvencí hodinových signálu [1]

Paměť by nedokázala procesoru při této frekvenci poskytnout data tak rychle. Z toho důvodu je nutné nastavit prodloužení čekací doby při čtení z paměti.

Po této změně, bez níž by kontroler nemohl správně fungovat, už stačí pouze zvolit fázový závěs jako zdroj systémového hodinového signálu a počkat, až k přepnutí dojde.

6.1.1.2 TIM

Nastavení časovače je v celku jednoduché. Nejprve jako vždy přivedeme do periferie hodinový signál.

Poté nastavíme konfigurační registry, děličku kmitočtu a Autoreload registr, tak, aby k přetečení došlo jednou za sekundu.

Povolíme přerušeni při přetečení, a poté i linku přerušeni jádra ARM, která je na časovač napojena.

```
void TIM2_init()
{
    // zapne hodinovy signal pro TIM2
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

    // nastavi delicku pro TIM2 (32 MHz / 32000 = 1 kHz)
    TIM2->PSC = 31999U;

    // nastavi autoreload registr pro TIM2 (1 kHz / 1000 = 1 Hz)
    TIM2->ARR = 999U;

    // povoli preruseni pri pretečení (update)
    TIM2->DIER |= TIM_DIER_UIE;

    // povoli preruseni linky ARM jadra pro TIM2
    NVIC->ISER[0] |= (1U << TIM2_IRQn);
}
```

6.1.2 Funkce periferií

6.1.2.1 TIM - start

Funkce pro spuštění časovače jednoduše nastaví povolovací bit v jeho řídicím registru.

```
void TIM2_start()
{
    // zapne TIM2
    TIM2->CR1 |= TIM_CR1_CEN;
}
```

6.1.3 Obsluha přerušení

Obsluha přerušení je také velmi snadná.

Pokud je zdrojem přerušení přetečení časovače, dojde k přepnutí stavu LED a vynulování příznaku přerušení.

```
void TIM2_IRQHandler(void)
{
    // preruseni z duvodu pretečení TIM2
    if(((TIM2->SR & TIM_SR_UIF) == TIM_SR_UIF) &&
        ((TIM2->DIER & TIM_DIER_UIE) == TIM_DIER_UIE))
    {
        LED_prepni();

        // vynuluj priznak preruseni
        TIM2->SR &= ~(TIM_SR_UIF);
    }
}
```

Teoreticky je v tomto příkladu podmínka zbytečná, jelikož jiný důvod přerušení není možný. Odstraněním podmínky kód zmenšíme a zrychlíme. Pro úplnost ji však do programu vložíme, kdybychom se jej v budoucnu rozhodli rozšířit.

6.1.4 Tvorba programu

Celý program vypadá následovně. Nejprve nastavíme zdroj hodinového signálu. Poté připravíme funkci LED na blikání. Následuje nastavení časovače a jeho spuštění. Nic víc není potřeba.

```
int main(void)
{
    RCC_init();

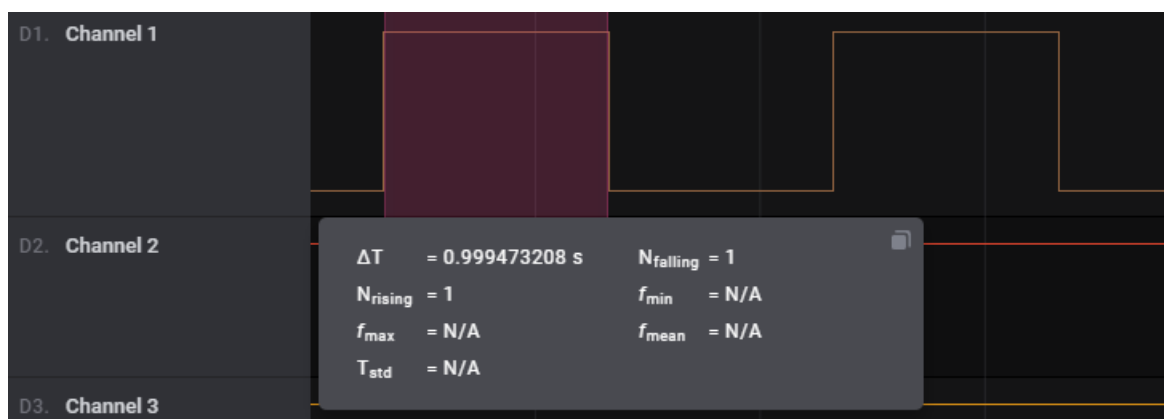
    LED_init();

    TIM2_init();

    TIM2_start();

    while(1);
}
```

Na následujícím obrázku je vidět, že doba svitu je skutečně jedna sekunda. Nepřesnost je dána tolerancí vnitřního oscilátoru.



6.1.5 Porovnání

Jelikož jsme v tomto příkladu pracovali s nastavením zdroje systémového hodinového signálu, není v porovnávacím programu s knihovnamy odstraněna inicializační funkce. To má velký dopad na výsledky v oblasti velikosti kódu.

Bez optimalizace			
Kmitočet hodinového signálu = 32 MHz	S knihovnamy	Bez knihoven	Úspora [%]
Velikost kódu [kB]	7,33	0,836	88,6
Obsluha přerušení přetečení [μs]	9,71	2,71	72,1
Optimalizace velikosti			
Velikost kódu [kB]	4,22	0,707	83,2
Obsluha přerušení přetečení [μs]	5,17	1,37	73,5
Optimalizace rychlosti			
Velikost kódu [kB]	4,99	0,668	86,6
Obsluha přerušení přetečení [μs]	4,79	1,50	68,7

Všimněme si také velkého rozdílu času obsluhy přerušení. Důvodem je komplexnost knihovní funkce, která dokáže reagovat na jakoukoliv příčinu přerušení.